Offline Policy-search in Bayesian Reinforcement Learning

Castronovo Michael

University of Liège, Belgium
Advisor: Damien Ernst

15th March 2017

Contents

Introduction

What is Reinforcement Learning (RL)?

A sequential decision-making process where an agent observes an environment, collects data and reacts appropriately.

Example: Train a Dog with Food Rewards

- Context: Markov-decision process (MDP)
- Single trajectory (= only 1 try)
- Discounted rewards (= early decisions are more important)
- Infinite horizon (= the number of decisions is infinite)

The Exploration / Exploitation dilemma (E/E dilemma)

An agent has two objectives:

- Increase its knowledge of the environment
- Maximise its short-term rewards

$\Rightarrow$ Find a compromise to avoid suboptimal long-term behaviour

In this work, we assume that

- The reward function is known
  (= agent knows if an action is good or bad)
- The transition function is unknown
  (= agent does not know how actions modify the environment)

Reasonable assumption:
Transition function is not unknown, but is instead uncertain:

$\Rightarrow$ We have some prior knowledge about it

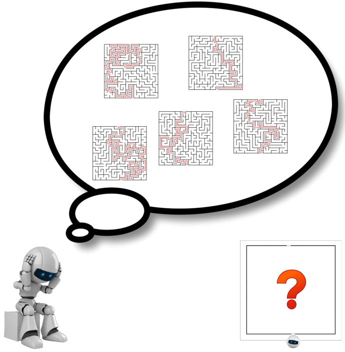
$\Rightarrow$ This setting is called Bayesian Reinforcement Learning

What is Bayesian Reinforcement Learning (BRL)?

A Reinforcement Learning problem where we assume some prior knowledge is available on start in the form of a MDP distribution.

Intuitively...

A process that allows to simulate decision-making problems similar to the one we expect to face.



Example:
A robot has to find the exit of an unknown maze.

→ Perform simulations on other mazes beforehand

→ Learn an algorithm based on those experiences (e.g.: *Wall follower*)

Contents

Problem statement

Let $M = (X, U, x_0, f_M(\cdot), \rho_M(\cdot), \gamma)$ be a given unknown MDP, where

- $X = \{x^{(1)}, \ldots, x^{(n_X)}\}$ denotes its finite state space
- $U = \{u^{(1)}, \ldots, u^{(n_U)}\}$ denotes its finite action space
- x_M^0 denotes its initial state.
- $x' \sim f_M(x, u)$ denotes the next state when performing action u in state x
- $r_t = \rho_M(x_t, u_t, x_{t+1}) \in [R_{\min}, R_{\max}]$ denotes an instantaneous deterministic, bounded reward
- $\gamma \in [0, 1]$ denotes its discount factor

Let $h_t = (x_M^0, u_0, r_0, x_1, \cdots, x_{t-1}, u_{t-1}, r_{t-1}, x_t)$ denote the history observed until time t.

An E/E strategy is a stochastic policy π that, given the current history h_t returns an action u_t:

$$u_t \sim \pi(h_t)$$

The expected return of a given E/E strategy π on MDP M:

$$J_M^\pi = \mathbb{E}_M \left[\sum_t \gamma^t r_t \right]$$

where

$$\begin{aligned} x_0 &= x_M^0 \\ x_{t+1} &\sim f_M(x_t, u_t) \\ r_t &= \rho_M(x_t, u_t, x_{t+1}) \end{aligned}$$

RL (no prior distribution)

We want to find a high-performance E/E strategy π^*_M for a given MDP M:

$$\pi^*_M \in \arg\max_{\pi} \; J^{\pi}_M$$

BRL (prior distribution $p^0_{\mathcal{M}}(\cdot)$)

A prior distribution defines a distribution over each uncertain component of $\mathcal{M}$ ($f_M(\cdot)$ in our case).

Given a prior distribution $p^0_{\mathcal{M}}(\cdot)$, the goal is to find a policy π^*, called *Bayes optimal*:

$$\pi^* = \arg\max_{\pi} \; \mathfrak{J}^{\pi}_{p^0_{\mathcal{M}}(\cdot)}$$

where

$$\mathfrak{J}^{\pi}_{p^0_{\mathcal{M}}(\cdot)} = \mathop{\mathbb{E}}_{M \sim p^0_{\mathcal{M}}(\cdot)} J^{\pi}_M$$

Contents

Offline Prior-based Policy-search (OPPS)

1. Define a rich set of E/E strategies:
   - $\rightarrow$ Build a large set of N formulas
   - $\rightarrow$ Build a formula-based strategy for each formula of this set
2. Search for the best E/E strategy in average, according to the given MDP distribution:
   - $\rightarrow$ Formalise this problem as an N-armed bandit problem

1. Define a rich set of E/E strategies

Let $\mathbb{F}^K$ be the discrete set of formulas of size at most K. A formula of size K is obtained by combining K elements among:

- Variables:
  $\hat{Q}_1^t(x,u),\ \hat{Q}_2^t(x,u),\ \hat{Q}_3^t(x,u)$
- Operators:
  $+,\ -,\ \times,\ /,\ |\cdot|,\ \frac{1}{\cdot},\ \sqrt{\cdot},\ \min(\cdot,\cdot),\ \max(\cdot,\cdot)$

Examples:

- Formula of size 2: $F(x,u) = |\,\hat{Q}_1^t(x,u)\,|$
- Formula of size 4: $F(x,u) = \hat{Q}_3^t(x,u) - |\,\hat{Q}_1^t(x,u)\,|$

To each formula $F \in \mathbb{F}^K$, we associate a formula-based strategy π_F, defined as follows:

$$\pi_F(h_t) \in \underset{u \in U}{\arg\max}\ F(x_t, u)$$

Problems:

- $\mathbb{F}^K$ is too large
  ($|\mathbb{F}^5| \simeq 300,000$ formulas for 3 variables and 11 operators)
- Formulas of $\mathbb{F}^K$ are redundant
  ($=$ different formulas can define the same policy)

  Examples:

  1. $Q_1^t(x,u)$ and $Q_1^t(x,u) - Q_3^t(x,u) + Q_3^t(x,u)$
  2. $Q_1^t(x,u)$ and $\sqrt{Q_1^t(x,u)}$

Solution:

$\Rightarrow$ Reduce $\mathbb{F}^K$

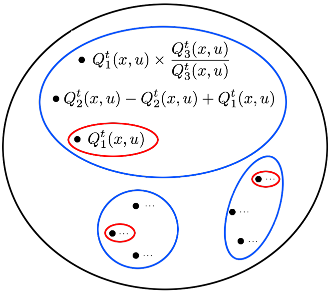
Reduction process

$\rightarrow$ Partition $\mathbb{F}^K$ into equivalence classes, two formulas being equivalent if and only if they lead to the same policy

$\rightarrow$ Retrieve the formula of minimal length of each class into a set $\bar{\mathbb{F}}^K$

Example:
$|\bar{\mathbb{F}}^5| \simeq 3,000$ while $|\mathbb{F}^5| \simeq 300,000$

Computing $\bar{\mathbb{F}}^K$ may be expensive. We instead use an efficient heuristic approach to compute a good approximation of this set.



2. Search for the best E/E strategy in average

A naive approach based on Monte-Carlo simulations (= evaluating all strategies) is time-inefficient, even after the reduction of the set of formulas.

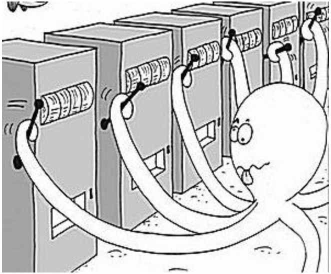
Problem:

In order to discriminate between the formulas, we need to compute an accurate estimation of $\mathfrak{J}^{\pi}_{p^0_{\mathcal{M}}(\cdot)}$ for each formula, which requires a large number of simulations.

Solution:

Distribute the computational ressources efficiently.
$\Rightarrow$ Formalise this problem as a multi-armed bandit problem and use a well-studied algorithm to solve it.

What is a multi-armed bandit problem?



A reinforcement learning problem where the agent is facing bandit machines and has to identify the one providing the highest reward in average with a given number of tries.

Formalisation

Formalise this research as a N-armed bandit problem.

- To each formula $F_n \in \bar{\mathbb{F}}^K$ ($n \in \{1, \dots, N\}$), we associate an arm
- Pulling the arm n consists in randomly drawing a MDP M according to $p^0_{\mathcal{M}}(\cdot)$, and perform a single simulation of policy π_{F^n} on M
- The reward associated to arm n is the observed discounted return of π_{F^n} on M

$\Rightarrow$ This defines a multi-armed bandit problem for which many algorithms have been proposed (e.g.: UCB1, UCB-V, KL-UCB, ...)

Learning Exploration/Exploitation in Reinforcement Learning

M. Castronovo, F. Maes, R. Fonteneau & D. Ernst (EWRL 2012, 8 pages)

BAMCP versus OPPS: an Empirical Comparison

M. Castronovo, D. Ernst & R. Fonteneau (BENELEARN 2014, 8 pages)

Contents

Artificial Neural Networks for BRL (ANN-BRL)

We exploit an analogy between decision-making and classification problems.

A reinforcement learning problem consists in finding a policy π which associates an action $u \in U$ to any history h.

A multi-class classification problem consists in finding a rule $\mathcal{C}(\cdot)$ which associates a class $c \in \{1, \ldots, K\}$ to any vector $v \in \mathbb{R}^n$ ($n \in \mathbb{N}$).

$\Rightarrow$ Formalise a BRL problem as a classification problem in order to use any classification algorithms such as Artificial Neural Networks

1. Generate a training dataset:
   - $\rightarrow$ Perform simulations on MDPs drawn from $p^0_{\mathcal{M}}(\cdot)$
   - $\rightarrow$ For each encountered history, recommend an action
   - $\rightarrow$ Reprocess each history h into a vector of fixed size
     - $\Rightarrow$ Extract a fixed set of features ($=$ variables for OPPS)
2. Train ANNs:
   - $\Rightarrow$ Use a boosting algorithm

1. Generate a training dataset

In order to generate a trajectory, we need a policy:

- A random policy?

  Con: Lack of histories for late decisions

- An optimal policy? ($f_M(\cdot)$ is known for $M \sim p^0_{\mathcal{M}}(\cdot)$)

  Con: Lack of histories for early decisions

$\Rightarrow$ Why not both?

Let $\pi^{(i)}$ be an ϵ-Optimal policy used for drawing trajectory i (on a total of n trajectories).

$$\text{For } \epsilon = \frac{i}{n} : \; \pi^{(i)}(h_t) = u^* \text{ with probability } 1 - \epsilon$$

and is drawn randomly in U else.

To each history $h_0^{(1)}, \ldots, h_{T-1}^{(1)}, \ldots, h_0^{(n)}, \ldots, h_{T-1}^{(n)}$ observed during the simulations, we associate a label to each action:

- 1 if we recommend the action
- -1 else

Example:

$U = \{u^{(1)}, u^{(2)}, u^{(3)}\}:\ h_0^{(1)} \leftrightarrow (-1,\ 1,\ -1)$
$\Rightarrow$ We recommend action $u^{(2)}$

We recommend actions which are optimal w.r.t. M
($f_M(\cdot)$ is known for $M \sim p^0_{\mathcal{M}}(\cdot)$).

Reprocess of all histories in order to fed the ANNs with vectors of fixed size.
$\Rightarrow$ Extract a fixed set of N features: $\phi_{h_t} = [\phi_{h_t}^{(1)}, \ \ldots, \ \phi_{h_t}^{(N)}]$

We considered two types of features:

- Q-Values:

$$\phi_{h_t} = [Q_{h_t}(x_t, u^{(1)}), \ \ldots, \ Q_{h_t}(x_t, u^{(n_U)})]$$

- Transition counters:

$$\begin{aligned}\phi_{h_t} = [C_{h_t}(< x^{(1)}, u^{(1)}, x^{(1)} >), \ \ldots, \\ C_{h_t}(< x^{(n_X)}, u^{(n_U)}, x^{(n_X)} >)]\end{aligned}$$

2. Train ANNs

Adaboost algorithm:

1. Associate a weight to each sample of the training dataset
2. Train a weak classifier on the weighted training dataset
3. Increase the weights of the samples misclassified by the combined weak classifiers trained previously
4. Repeat from Step 2

Problems

- Adaboost only addresses two-class classification problems (reminder: we have one class for each action)
  $\Rightarrow$ Use SAMME algorithm instead
- Backpropagation does not take the weights of the samples into account
  $\Rightarrow$ Use a re-sampling algorithm for the training dataset

Approximate Bayes Optimal Policy Search using NNs

M. Castronovo, V. François-Lavet, R. Fonteneau, D. Ernst & A. Couëtoux (ICAART 2017, 13 pages)

Contents

Benchmarking for BRL

Bayesian litterature

Compare the performance of each algorithm on well-chosen MDPs with several prior distributions.

Our benchmark

Compare the performance of each algorithm on a distribution of MDPs using a (possibly) different distribution as prior knowledge.

Prior distribution $=$ Test distribution $\Rightarrow$ Accurate case

Prior distribution $\neq$ Test distribution $\Rightarrow$ Inaccurate case

Additionally, computation times of each algorithm is part of our comparison criteria.

Motivations:

- $\Rightarrow$ No selection bias
(= good on a single MDP $\neq$ good on a distribution of MDPs)
- $\Rightarrow$ Accurate case evaluates generalisation capabilities
- $\Rightarrow$ Inaccurate case evaluates robustness capabilities
- $\Rightarrow$ Real-life applications are subject to time constraints
(= computation times cannot be overlooked)

The Experimental Protocol

An experiment consists in evaluating the performances of several algorithms on a test distribution $p_{\mathcal{M}}(\cdot)$ when trained on a prior distribution $p^0_{\mathcal{M}}(\cdot)$.

One algorithm $\rightarrow$ several agents (we test several configurations)

We draw N MDPs $M^{(1)}, \ldots, M^{(N)}$ from the test distribution $p_{\mathcal{M}}(\cdot)$ in advance, and we evaluate the agents as follows:

- $\rightarrow$ Build policy π offline w.r.t. $p^0_{\mathcal{M}}(\cdot)$
- $\rightarrow$ For each sampled MDP $M^{(i)}$, compute estimate $\bar{J}^{\pi}_{M^{(i)}}$ of $J^{\pi}_{M^{(i)}}$
- $\rightarrow$ Use these values to compute estimate $\bar{\mathfrak{J}}^{\pi}_{p_{\mathcal{M}}(\cdot)}$ of $\mathfrak{J}^{\pi}_{p_{\mathcal{M}}(\cdot)}$

Estimate J^π_M:
Truncate each trajectory after T steps:

$$\eta = 0.001$$
$$T = \left\lceil \frac{\log(\eta \times (1-\gamma))}{R_{max}} / \log \gamma \right\rceil$$
$$J^\pi_M \approx \bar{J}^\pi_M = \sum_t^T r_t \gamma^t$$

where η denotes the accuracy of our estimate.

Estimate $\mathfrak{J}^{\pi}_{p_{\mathcal{M}}(\cdot)}$:

We compute $\mu_\pi = \bar{\mathfrak{J}}^{\pi}_{p_{\mathcal{M}}(\cdot)}$ and σ_π, the empirical mean and standard deviation of the results observed on the N MDPs drawn from $p_{\mathcal{M}}(\cdot)$.

The statistical confidence interval at 95% for $\mathfrak{J}^{\pi}_{p_{\mathcal{M}}(\cdot)}$ is computed as:

$$\mathfrak{J}^{\pi}_{p_{\mathcal{M}}(\cdot)} \approx \bar{\mathfrak{J}}^{\pi}_{p_{\mathcal{M}}(\cdot)} = \frac{1}{N} \sum_{1 \leq i \leq N} \bar{J}^{\pi}_{M^{(i)}}$$

$$\mathfrak{J}^{\pi}_{p_{\mathcal{M}}(\cdot)} \in \left[\bar{\mathfrak{J}}^{\pi}_{p_{\mathcal{M}}(\cdot)} - \frac{2\sigma_\pi}{\sqrt{N}};\ \bar{\mathfrak{J}}^{\pi}_{p_{\mathcal{M}}(\cdot)} + \frac{2\sigma_\pi}{\sqrt{N}} \right]$$

Time constraints

We want to classify algorithms based on their time performance.

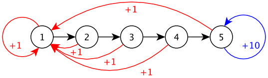
More precisely, we want to identify the best algorithm(s) with respect to:

1. Offline computation time constraint
2. Online computation time constraint

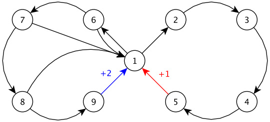
We filter the agents depending on the time constraints:

- Agents not satisfying the time constraints are discarded
- For each algorithm, we select the best agent in average
- We build the list of agents whose performances are not statistically different than the best one observed (Z-test)

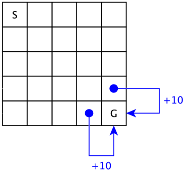
Experiments



GC - Generalised Chain



GDL - Generalised Double-loop



Grid

$\text{GC}(n_x = 5, n_U = 3)$; $\text{GDL}(n_x = 9, n_U = 2)$; $\text{Grid}(n_x = 25, n_U = 4)$

Simple algorithms

- Random
- ϵ-Greedy
- Soft-Max

State-of-the-art BRL algorithms

- BAMCP
- BFS3
- SBOSS
- BEB

Our algorithms

- OPPS-DS
- ANN-BRL

Results

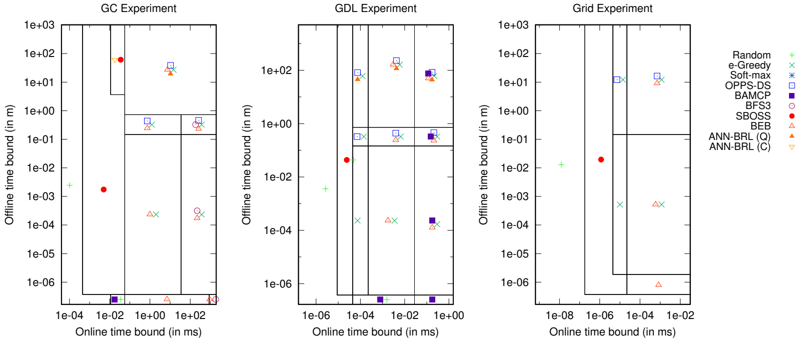


Figure: Best algorithms w.r.t offline/online periods (accurate case)

Agent	Score on GC	Score on GDL	Score on Grid
Random	31.12 ± 0.90	2.79 ± 0.07	0.22 ± 0.06
e-Greedy	40.62 ± 1.55	3.05 ± 0.07	6.90 ± 0.31
Soft-Max	34.73 ± 1.74	2.79 ± 0.10	0.00 ± 0.00
BAMCP	35.56 ± 1.27	$\mathbf{3.11 \pm 0.07}$	6.43 ± 0.30
BFS3	39.84 ± 1.74	2.90 ± 0.07	3.46 ± 0.23
SBOSS	35.90 ± 1.89	2.81 ± 0.10	4.50 ± 0.33
BEB	41.72 ± 1.63	3.09 ± 0.07	6.76 ± 0.30
OPPS-DS	$\mathbf{42.47 \pm 1.91}$	3.10 ± 0.07	$\mathbf{7.03 \pm 0.30}$
ANN-BRL (Q)	42.01 ± 1.80	$\mathbf{3.11 \pm 0.08}$	6.15 ± 0.31
ANN-BRL (C)	35.95 ± 1.90	2.81 ± 0.09	4.09 ± 0.31

Table: Best algorithms w.r.t Performance (accurate case)

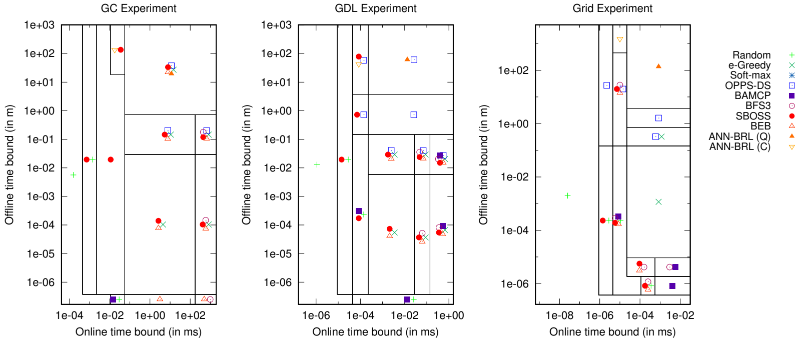


Figure: Best algorithms w.r.t offline/online periods (inaccurate case)

Agent	**Score on GC**	**Score on GDL**	**Score on Grid**
Random	31.67 ± 1.05	2.76 ± 0.08	0.23 ± 0.06
e-Greedy	37.69 ± 1.75	2.88 ± 0.07	0.63 ± 0.09
Soft-Max	34.75 ± 1.64	2.76 ± 0.10	0.00 ± 0.00
BAMCP	33.87 ± 1.26	2.85 ± 0.07	0.51 ± 0.09
BFS3	36.87 ± 1.82	2.85 ± 0.07	0.42 ± 0.09
SBOSS	38.77 ± 1.89	2.86 ± 0.07	0.29 ± 0.07
BEB	38.34 ± 1.62	2.88 ± 0.07	0.29 ± 0.05
OPPS-DS	$\mathbf{39.29 \pm 1.71}$	$\mathbf{2.99 \pm 0.08}$	1.09 ± 0.17
ANN-BRL (Q)	38.76 ± 1.71	2.92 ± 0.07	$\mathbf{4.29 \pm 0.22}$
ANN-BRL (C)	36.30 ± 1.82	2.84 ± 0.08	0.91 ± 0.15

Table: Best algorithms w.r.t Performance (inaccurate case)

BAMCP versus OPPS: an Empirical Comparison

M. Castronovo, D. Ernst & R. Fonteneau (BENELEARN 2014, 8 pages)

Benchmarking for Bayesian Reinforcement Learning

M. Castronovo, D. Ernst, A. Couëtoux & R. Fonteneau (PLoS One 2016, 25 pages)

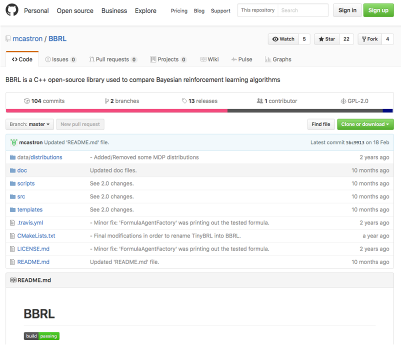
Contents

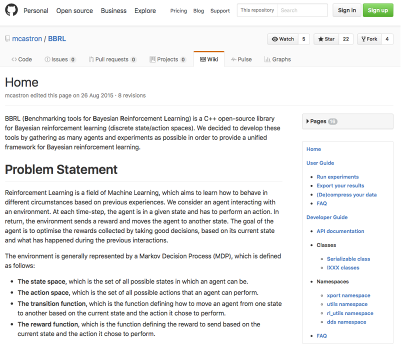
Conclusion

Summary

1. **Algorithms:**
   - Offline Prior-based Policy-search (OPPS)
   - Artificial Neural Networks for BRL (ANN-BRL)
2. **New BRL benchmark**
3. **An open-source library**

BBRL: **B**enchmarking tools for **B**ayesian **R**einforcement **L**earning





https://github.com/mcastron/BBRL/

Future work

- OPPS
  - $\rightarrow$ Feature selection (PCA)
  - $\rightarrow$ Continuous formula space
- ANN-BRL
  - $\rightarrow$ Extension to high-dimensional problems
  - $\rightarrow$ Replace ANNs by other ML algorithms (e.g.: SVMs, decision trees)
- BRL Benchmark
  - $\rightarrow$ Design new distributions to identify specific characteristics
- Flexible BRL algorithm
  - $\rightarrow$ Design an algorithm to exploit both offline and online phases

Thanks for your attention!